

Algorithmique 1

Durée : 3h

Machines électroniques interdites

Tous documents papiers autorisés

Les deux parties du sujet sont indépendantes. Veuillez rendre ces deux parties sur des feuilles séparées.

Veuillez respecter les notations introduites dans l'énoncé. Il est inutile de paraphraser l'énoncé dans vos réponses, mais des explications avec dessins sur votre code sont les bienvenues.

Dans l'ensemble de cet examen, on travaille avec le type `Liste` des listes simplement chaînées, introduit par les définitions suivantes :

```
type Cellule;  
  
type Liste is access Cellule;  
  
type Cellule is record  
  Val : Integer;  
  Suiv : Liste;  
end record;  
  
procedure Liberer is new Ada.Unchecked_Deallocation (Cellule, Liste);
```

FIGURE 1 – Définition du type `Liste`

```

package FilePrio is

  subtype Elt is Integer range ...;

  function EstVide return Boolean;
  -- retourne True ssi la file ne contient aucun élément.

  procedure Vider;
  -- garantit EstVide = True

  procedure Insérer (X : in Elt);
  -- garantit une nouvelle occurrence de X est insérée dans la file.

  procedure DetruireMax (Max : out Elt);
  -- requiert EstVide = False
  -- garantit une occurrence de Max est retirée de la file,
  -- et Max est >= à tous les éléments restants dans la file.

end FilePrio;

```

FIGURE 2 – Interface de FilePrio

En supposant $\text{Elt}'\text{First} \leq 3$ et $10 \leq \text{Elt}'\text{Last}$, on considère la séquence ci-contre.

A l'issue de cette séquence, la variable X vaut 10. La file de priorités contient ici les éléments suivants : “3” en deux exemplaires, “5” en quatre exemplaires, et “10” en un exemplaire.

Si on fait ensuite un nouvel appel à `DetruireMax (X)`, la variable X vaut encore 10, mais la file ne contient plus l'élément 10. Le maximum de la file est donc 5 (en quatre exemplaires).

```

X : Elt;
begin
  ...
  Vider;
  Insérer (3);
  Insérer (10);
  Insérer (5);
  Insérer (5);
  Insérer (10);
  Insérer (5);
  Insérer (3);
  Insérer (5);
  DetruireMax (X);

```

FIGURE 3 – Exemple d'utilisation de FilePrio

L'implémentation par tableau utilise les deux variables globales :

```

NbOcc : array (Elt) of Natural;
MaxFile : Elt;

```

L'invariant satisfait par ces deux variables du paquetage est la conjonction des deux propriétés :

1. si $\text{NbOcc} = (\text{others} \Rightarrow 0)$ alors $\text{MaxFile} = \text{Elt}'\text{First}$;
2. si $\text{NbOcc} \neq (\text{others} \Rightarrow 0)$ alors $\text{NbOcc}(\text{MaxFile}) \neq 0$ et pour tout $I : \text{Elt}$ tel que $\text{NbOcc}(I) \neq 0$, on a $I \leq \text{MaxFile}$.

FIGURE 4 – Variables globales de FilePrio dans implém. par tableau

I - Files de priorités (12 points)

L'objectif de cet exercice est d'étudier différentes implémentations d'une structure de données appelée *file de priorités*. Les implémentations abordées ici sont relativement naïves. Des implémentations plus efficaces seront traitées au deuxième semestre.

On travaille ici avec un paquetage `FilePrio` dont l'interface est donnée figure 2 page 2. L'état interne de ce paquetage représente une suite ordonnée d'éléments, appelée *file de priorités*. Ces éléments sont du type `Elt`, un sous-type intervalle de `Integer`. L'ordre sur les éléments est donc l'ordre par défaut sur `Integer`. Le mode d'emploi du paquetage précise que l'état initial (au chargement du programme) n'est pas défini. Il faut donc commencer par appeler `Vider` avant d'utiliser le reste du paquetage. La figure 3 page 2 donne un exemple d'utilisation du paquetage. Dans la suite, on note N le cardinal de `Elt` (le nombre d'éléments de l'intervalle `Elt'Range`).

▷ Question 1 (1 point) :

Utiliser ce paquetage pour définir une procédure `Tri` qui ordonne les éléments d'un tableau `T` suivant l'ordre décroissant.

```
type Tab is array (Integer range <>) of Elt;  
procedure Tri (T : in out Tab);
```

1 Implémentation par un tableau

Dans un premier temps, on suppose que N est relativement petit. On représente la file par un tableau `NbOcc` qui associe à chaque élément son nombre d'occurrences (ou d'exemplaires) dans la suite. Voir figure 4 page 2. Par exemple, à l'issue de la suite d'instructions donnée à la figure 3, on a :

`NbOcc = (3 => 2, 5 => 4, 10 => 1, others => 0)`

Pour rendre `DetruireMax` plus efficace, on calcule à l'avance le maximum courant de la file dans une variable globale appelée `MaxFile`. Typiquement pour le `NbOcc` de l'exemple précédent, `MaxFile=10`. Plus précisément, ces 2 variables doivent satisfaire l'invariant du paquetage donné figure 4 page 2 : si la file est vide, `MaxFile` vaut `Elt'First`, sinon il vaut l'élément maximum de la file.

▷ Question 2 (4 points) :

Implémenter les sous-programmes du paquetage en optimisant les temps de calculs autant que possible.

1. Écrire `Vider` (0,5 point) : cette procédure doit établir l'invariant.
2. Écrire `EstVide` (0,5 point) en utilisant l'invariant : le coût en temps de cette fonction est constant en fonction de N .
3. Écrire `Inserer` (1 point) : cette procédure doit préserver l'invariant.
4. Écrire `DetruireMax` (1,5 points) : cette procédure doit préserver l'invariant.

2 Deux implémentations par une liste chaînée

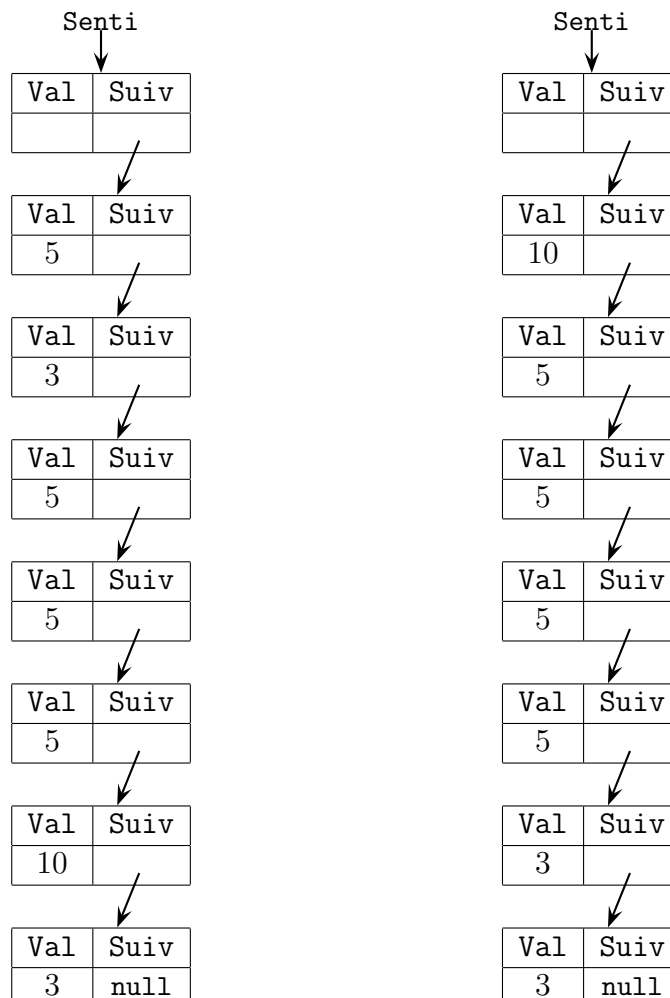
Maintenant, on étudie deux autres implémentations du paquetage, qui peuvent être intéressantes par rapport à l'implémentation précédente lorsque N est grand et que le nombre d'occurrences de chaque éléments de la file reste faible. Dans ces deux implémentations, le paquetage n'a qu'une seule variable globale interne qui est une liste simplement chaînée de type `Liste` (défini figure 1 page 1). Cette liste a une sentinelle de tête pour rendre les algorithmes réguliers. Dans votre code, vous devez utiliser cette sentinelle, et éviter des analyses par cas inutiles.

Cette sentinelle est allouée à l'initialisation du paquetage et elle n'est jamais libérée. Cette variable est donc simplement appelée `Senti` et déclarée comme ci-dessous dans le paquetage :

```
Senti : Liste := new Cellule;
```

Dans les deux implémentations, l'invariant du paquetage garantit que `Senti` est une liste simplement chaînée **non vide** et bien formée (sans cycle et sans pointeur pendante), et que les éléments de la file sont ceux accessibles depuis la liste de tête `Senti.Suiv`.

Par exemple, les deux listes chaînées dessinées ci-dessous peuvent représenter la file obtenue à l'issue de la séquence donnée figure 3. Dans cette figure le champ `Val` de la cellule pointée par `Senti` est laissée vide pour signifier que sa valeur n'est jamais utilisée.



▷ **Question 3 (1 point) :**

Programmer `EstVide` et `Vider` en respectant le fait que `Senti` est une sentinelle de tête. la procédure `Vider` doit libérer les éventuelles cellules devenant inutilisées.

2.1 Première implémentation

Dans la première des 2 implémentations utilisant `Senti`, on ne contraint pas davantage l'invariant sur `Senti`.

▷ **Question 4 (1 point) :**

Écrire la procédure `Inserer` de sorte qu'elle place la nouvelle occurrence de `X` en tête de liste (dans le suivant de la sentinelle).

▷ **Question 5 (1 point) :**

Écrire une fonction interne au paquetage, appelée `CherchePredMax` qui a la spécification suivante :

```
function CherchePredMax return Liste is
-- requiert la file contient au moins un élément.
-- retourne un pointeur P non nul accessible depuis Senti
--     tel que le suivant de P est non nul
--     et la valeur de la cellule suivant P
--     est le max des éléments de la file.
```

▷ **Question 6 (1 point) :**

Utiliser `CherchePredMax` pour écrire `DetruireMax`. La procédure `DetruireMax` doit libérer la cellule qui n'est plus utilisée.

2.2 Deuxième implémentation

Dans cette dernière implémentation, on contraint maintenant l'invariant sur `Senti` à garantir que la liste dont la tête est `Senti.Suiv` est triée par ordre décroissant sur les éléments (comme sur le dessin de droite, page 4).

▷ **Question 7 (2 points) :**

Reprogrammer `Inserer` et `DetruireMax` pour respecter ce nouvel invariant. La procédure `DetruireMax` doit libérer la cellule qui n'est plus utilisée.